Reinventing CS50

Reinventing CS50

David J. Malan
Harvard University
School of Engineering and Applied Sciences
Cambridge, Massachusetts, USA
malan@post.harvard.edu

ABSTRACT

Computer Science 50 is Harvard College's introductory course for majors and non-majors alike, enrollment in which both rose and fell along with the dotcoms. Although enrollment peaked in 1996 at 386 students, it had settled by 2002 in the neighborhood of 100.

We set out in 2007 to combat that trend by tackling two problems. We hypothesized that CS50 suffered from two, one of perception and one of design. Although, per end-of-term surveys, the course had never lacked for good teachers or good content, the consensus on campus for years had been to beware this particular course. And the course's own syllabus may very well have been dated in the eyes of students who had begun to carry regularly modern hardware and software in their backpacks and pockets.

Not only did we proceed to revamp every one of CS50's problem sets, we brought its syllabus more in line with technological trends already familiar to students. And we altered the tone of the course to appeal to those "less comfortable" with computing on campus. But we took care to preserve the course's rigor and underlying fundamentals, lest we do our own students a disservice.

Our new approach appears to be working. Between 2006 and 2007, enrollment in CS50 more than doubled from 132 to 282 (+114%). Between 2007 and 2008, enrollment increased another 17% to 330, though even more striking was that year's 48% increase in female enrollment. By 2009, enrollment remained strong at 338.

We present in this work what we have done and why we have done it.

Categories and Subject Descriptors

K.3.2 [**COMPUTERS AND EDUCATION**]: Computer and Information Science Education—*Computer science education*; K.3.2 [**COMPUTERS AND EDUCATION**]: Computer and Information Science Education—*Curriculum*

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

General Terms

Design, Experimentation, Human Factors

Keywords

CS0, CS1, CS2, curriculum, pedagogy

1. INTRODUCTION

Computer Science 50 is Harvard College's "introduction to the intellectual enterprises of computer science and the art of programming" for majors and non-majors alike, a one-semester amalgam of courses generally known as CS1 and CS2. Although enrollment in CS50 spiked to 386 in 1996 (on a campus of 6500), our numbers, like most universities', not only rose but also fell with the dotcoms.[1] In 2002, enrollment dipped below 100 and, by 2006, had settled at 132. Of course, ten years prior, our numbers were in precisely that neighborhood too. What might, at first glance, have been a worrisome decline may, in fact, have been mere restoration of equilibrium.

But clearly there was an audience for computer science out there, even though external forces may have coaxed some members our way. And so we asked ourselves in 2007 whether we could recreate the excitement that the dotcoms instilled in so many students and whether we could attract similar numbers with forces more internal than external. To be sure, some of that era's students might have enrolled for academically wrong reasons (*e.g.*, dreams of getting rich quick). So we did not necessarily want all of them back. But among those 386 were some good computer scientists, some of whose potential might not have ever been realized had they not been coaxed from other departments. We wanted to find precisely those students in 2007.

We hypothesized that CS50 suffered from two problems, one of perception and one of design. Although the course was highly regarded, most everyone on campus seemed to think that it should only be taken with caution, not unlike similar courses elsewhere. Opinions were mixed on whether the cause for concern was the course's workload or difficulty, but the result was the same. The consensus was to beware. And it probably did not help that we began each semester from the ground up, introducing students first to assembly language and later to C. Not only does "hello, world" tend to underwhelm students who now carry iPhones and iPods, the syntax of these (and most languages, for that matter, Java

[1]We were fortunate in 1996 to have Brian Kernighan at the helm of CS50.

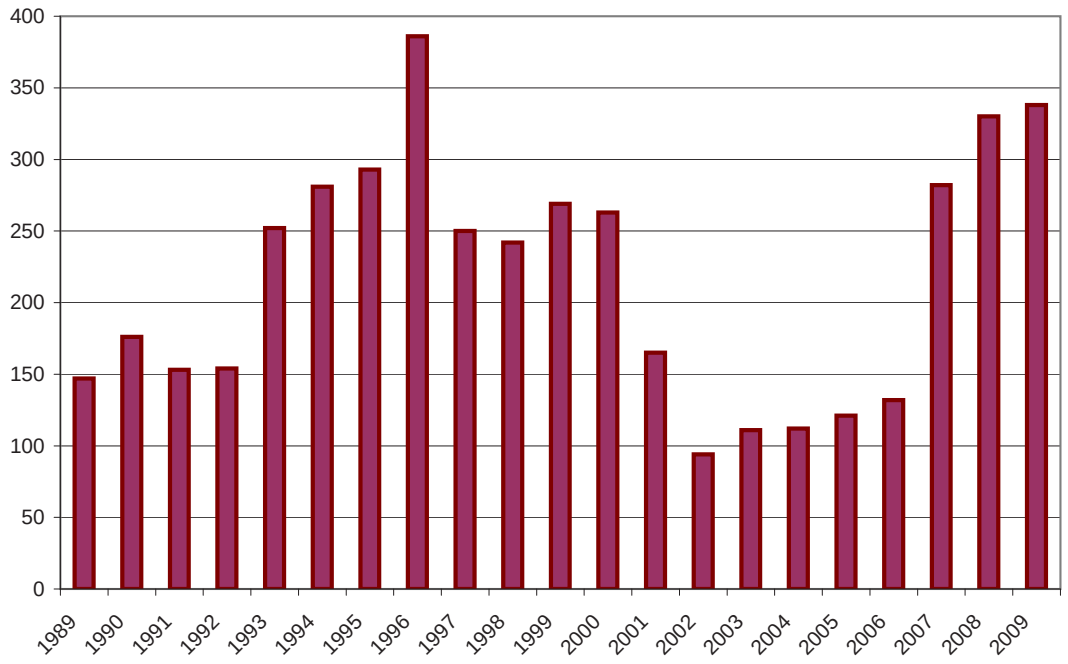

Figure 1: Enrollments in CS50 both rose and fell along with the dotcoms. But they have since risen again in lockstep with changes we began to implement in 2007. Between 2006 and 2007, enrollment increased 114% from 132 to 282. Between 2007 and 2008, enrollment increased another 17% to 330, though more striking was that year's 48% increase in female enrollment. By 2009, enrollment remained strong at 338.

included) tends to distract students from more interesting fundamentals in an introductory course's first weeks.

And so we set out to tackle both problems at once. Not only did we reorganize our syllabus, we modernized our problem sets, tying each one to a real-world domain more familiar to today's students. And we altered the tone of the course, embracing in Week 0 those "less comfortable" and "more comfortable" (with computing) alike. In a word, we set out to make the course more "accessible," while still preserving its content and rigor. We daresay it is easy to create a popular gut, but we had no such intention. Although CS50 is a terminal course for many non-majors, it is a gateway to higher-level courses for majors and minors and must still, therefore, provide no less strong a foundation. More "accessible" has not meant "easier" or "less work" but, rather, more friendly to those who might otherwise worry that they do not belong.

Thus far, our approach appears to be working, per Figure 1. Between 2006 and 2007, enrollment in CS50 increased 114% from 132 to 282. Between 2007 and 2008, enrollment increased another 17% to 330, though even more striking was that year's 48% increase in female enrollment. By 2009, enrollment remained strong at 338. In total, CS50's enrollment has increased by 156% since 2007. To be sure, enrollments in computer science programs have been on the rise nationally since 2007, but not by nearly as much (6.2%) [7].

We present in this paper the new CS50. In the section that follows, we offer background on the students who have begun to take this particular course. In Section 3, we discuss CS50's current philosophy and motivation for the same. In Section 4, we detail the course's curriculum. In Section 5, we survey the problem sets we have modernized. In Section 6, we present our results and, in Section 7, we conclude. Although we happen to spend most of CS50 in C, we suspect the lessons we've learned are no less applicable to courses in Java and other high-level languages.

2. STUDENT BODY

Although some students who take CS50 have one or more prior courses under their belt (*e.g.*, AP Computer Science), most of them (72%) do not. When asked to describe their level of comfort with computing (or the mere idea of being in CS50), 34% of Fall 2008's students described themselves as among those "less comfortable," 14% described themselves as among those "more comfortable," and 52% described themselves as somewhere in between.

Although more than half of the course tends to be male, female enrollment appears to be on the rise. Whereas female students composed 29% of Fall 2007's student body, they composed 36% of Fall 2008's thanks to an increase of 48% in raw numbers. In fact, almost all growth in enrollment between 2007 and 2008, from 282 to 330, was the result of more female students in 2008.

Although students gather twice weekly for large lectures, CS50 implements apprenticeship learning [1,2], whereby each student is apprenticed at term's start to a teaching fellow (TF) who grades that student's work and leads a weekly "section" (*i.e.*, recitation) for that student and a dozen or so others. Almost all of the course's 30 TFs are still undergraduates themselves who took CS50 one or more years prior.

Figure 2: CS50 is known for its workload. Most students spent at least 10 hours outside of class on each week's problem set, with some spending as many as 20.

3. PHILOSOPHY

It is not our intention in CS50 to weed students out but, rather, to open as many eyes as possible to a field we ourselves love. In exchange for their time, the course promises to get every student from Week 0 to Week 12. And the course's own syllabus assures them that "what ultimately matters in this course is not so much where you end up relative to your classmates but where you, in Week 12, end up relative to yourself in Week 0." We provide students with every resource they need to succeed, including lectures and sections (and videos thereof), in-person office hours, virtual office hours [4], code walkthroughs for problem sets (*i.e.*, programming assignments), an anonymized bulletin board, scribe notes, and more. Not only do these resources provide a support structure that empowers students to tackle each week a problem set that demands upwards of 15 hours of work outside of class, per Figure 2, they allow students to engage with the course from any number of angles. Students for whom lectures move quickly can review videos at home. Students who learn best in one-on-one contexts can find precisely that structure at office hours. And students who fear asking the proverbial dumb question can turn to the bulletin board and ask it anonymously. Sections, moreover, are offered in three different flavors: some for those "less comfortable," some for those "more comfortable," and some for those somewhere in between.

But the course plays as hard as it works students. We introduce students in lectures and problem sets not only to computer science but also to geek culture. Lolcats [6] exist alongside linked lists. Internet memes break the ice at classes' start. And YouTube clips allow students to breathe during what might otherwise be 90 intense minutes of lecture.

The course takes its curriculum seriously but not itself too much so. Of course, years from now, we may very well grimace at our conflation of science and geekery. But we think the rewards worth those particular risks. To date, this course is renowned as having one of the heaviest workloads at Harvard. But this same course is nonetheless now Harvard's fifth largest. There's much to be said for that spoonful of sugar. We expect much of our students (cf. Figure 2) in return for the fun that we have. And we get it.

4. CURRICULUM

We discuss in this section the overall arc of the course, including the topics (italicized) covered in lectures each week. Omitted below are weeks with vacations or quizzes. The course covers an unusual number of topics, some in more detail than others. The course's breadth is accompanied by depth in a subset of topics. But it's worth noting that we rely on problem sets to reinforce, hands-on, the topics we feel most important to students' foundation, per Section 5. Lectures are not the course's sole delivery mechanism for content.

4.1 Week 0

Introduction. Bits. Binary. ASCII. Programming. Algorithms. Scratch. Statements. Boolean expressions. Conditions. Loops. Variables. Threads. Events.

Though everyone's favorite canonical program, it's perhaps fair to say that "hello, world" fails to excite most students today. And converting Fahrenheit to Celsius is not terribly interesting. But it's hard to compete in Week 0 with the hardware and software that students carry in backpacks and pockets. There is only so much one can do with C's (or Java's) most basic syntax before it is time to introduce more sophisticated constructs (*e.g.*, arrays). Implementation (and comprehension) of GUIs is generally weeks, if not semesters, away. And so we do not use C in CS50's first week. We use instead Scratch [5], a drag-and-drop programming language developed by MIT's Media Lab, similar in spirit to Alice [3] *et al.*, but whose learning curve is much lower. Although designed for youth in after-school programs, we use Scratch to excite our students in Week 0. With it can they implement their own animations, games, and interactive art without the frustration of semicolons and other syntactical distractions that just aren't interesting in the first days of a course. And yet among Scratch's "puzzle pieces" are precisely the constructs we want to introduce in this first week of class: statements, Boolean expressions, conditions, loops, and variables. Scratch even allows us to discuss threads and events, topics generally reserved for non-introductory courses.

But we spend just one lecture and one problem set on Scratch, after which we immediately transition to C in Week 1.

4.2 Week 1

C. Source code. Compilers. Object code. SSH. SFTP. GCC. Functions. Comments. Standard output. Arithmetic operators. Precedence. Associativity. Local variables. Types. Casting. Standard input. Libraries. Boolean expressions, continued. Conditions, continued. Loops, continued.

In Week 1 do we dive head-first into C, but we feel comfortable waving our hand at certain syntactic details that might otherwise bog down the course's momentum. Yes, `for` loops are cryptic at first glance, but, thanks to Scratch, students already have a mental model into which they can fit that new "piece." We daresay the parentheses and semicolons intimidate them less because the underlying idea is already familiar. And so we spend this week discussing more interesting topics like standard input and output, functions and libraries, and Linux itself. We take care during lecture, though, to provide students with dozens of bite-sized examples with which to pick up C's particular syntax.

4.3 Week 2

Functions, continued. Global variables. Parameters. Return values. Stack. Frames. Scope. Arrays. Strings. Command-line arguments. Cryptography.

In Week 2, we continue to look at more sophisticated topics, including parameters and return values, stack frames and scope, and even arrays. But we also introduce the first of several real-world domains, namely cryptography. That particular topic allows us to challenge students this week to implement a number of ciphers as well as crack passwords using heuristics of their choice. Both activities reduce to some standard input and output, some loops, and some typecasting between `char`s and `int`s, all quite accessible to students at this point in the course. But the problem domain is intellectually "sexy," and that is what motivates.

4.4 Week 3

Linear search. Binary search. Asymptotic notation. Recursion. Pseudorandomness. Bubble sort. Selection sort. Insertion sort. Merge sort. Debugging.

In Week 3 do we transition away from syntax and details specific to C, instead elevating the discussion to algorithms and data structures. We discuss searching and sorting. We discuss efficiency and measurement thereof, including O, Θ, and Ω. Though, perhaps most importantly, we have students act out each of the algorithms. Students remember that bubble sort is slow after they've seen it acted out (not only inefficiently but awkwardly too) by volunteers on stage. We then reinforce those visuals with animations that contrast the algorithms' run times side by side.

4.5 Week 4

Structures. Dynamic memory allocation. Stack and heap. Pointers. Debugging, continued.

In Week 4 do we then discuss how to implement those algorithms in memory. And we discuss memory management in C as well as pointers, the latter of which absolutely takes time to sink in. And so we revisit that particular topic throughout the following weeks.

4.6 Week 5

File I/O. Forensics. Linked lists. Stacks. Queues.

We next introduce students to some file I/O so that their problem sets can finally read from and write to actual disks. We introduce yet another problem domain, this time digital forensics. And we introduce a handful of new structures, including linked lists.

4.7 Week 7

Valgrind. Bitwise operators. Hash tables. Trees. Binary search trees. Tries. Huffman coding.

In Week 7, we introduce students to more sophisticated data structures still, among them hash tables and tries. And, per Section 5, we don't have them implement those structures for the sake of the structures themselves but, rather, to solve more interesting problems.

4.8 Week 8

HTTP. XHTML. PHP. SQL.

In Week 8, we transition away from C altogether and begin a brief look at Web programming. We present the basics of XHTML and then move on to PHP, whose syntax is similar enough to C that we spend most of our time on the features it boasts over C. Moreover, `php.net` offers what may very well be the best documentation of any language today. It certainly facilitates students' picking up a language as of yet unfamiliar to them. We also familiarize students in this week with SQL, including `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, as well as the basics of schema design, but only enough so that they have an ability to store and retrieve data for the week's problem set.[2]

Our motivation for PHP in this week is twofold. Not only do we want to empower students to implement, at term's end, final projects that just so happen to be Web-based, we want them to realize that they are not taking a course about C but, rather, about programming and computer science itself. We want them to experience, with training wheels still on, how one goes about learning new languages, lest they assume, to their detriment, that they must wait for some subsequent course to teach them an additional language.

Other languages might fulfill these same goals. But we happen to like PHP, if only because `php.net` is such a good teacher.

4.9 Week 9

CSS. JavaScript. Events, continued. Ajax.

In Week 9, we both continue and conclude our look at Web programming, introducing students to DOM (yet another application of data structures already covered earlier in term) as well as to JavaScript, whose syntax is quite similar to PHP's own. Thanks to both are yet more possibilities for final projects accessible to students.

4.10 Week 10

Preprocessing. Compiling. Assembling. Linking. CPUs.

In Week 10 do we tease students with what's been underneath the hood all term long. We discuss what it really means to compile, assemble, and link code. And we discuss what "Intel inside" has meant all along.

5. PROBLEM SETS

But it is by way of the course's problem sets that the course's lessons truly sink in. We present in this section the challenges students now take on in each week.

Problem Set 0 invites students to implement most any Scratch project of interest to them, subject only to a few constraints. The aim of this first problem set is to get students excited, to do with Scratch what "hello, world" simply cannot. But the mere act of designing their own animation, game, or interactive art exposes them to statements, Boolean expressions, conditions, loops, and more, all of which prove fairly intuitive when so easily dragged and dropped.

Problem Set 1 next introduces students to Linux. It challenges students to implement a handful of bite-sized C programs, among them validation of ISBNs or credit cards, greedy change-making, and Super Mario's pyramid.

Problem Set 2 then presents that first real-world domain: cryptography. Students must either implement a pair of old ciphers (Caesar and Vigenère) or crack a copy of `/etc/passwd`.

Problem Set 3 is the first that provides students with skeletal code (for the Game of Fifteen) that they must first understand and whose blanks they must proceed to fill in.

[2]We provide each student with a MySQL database that they manage with phpMyAdmin, whose user-friendly GUI lowers the bar considerably to managing tables.

Problem Set 4 hands students even more skeletal code (over 600 lines) for Sudoku, whose implementation they must finish. This problem set also introduces students to APIs, in this case's ncurses's.

Problem Set 5 then has student dabble in the domain of forensics. We first stroll about campus with a digital camera, taking photos of indentifiable but non-obvious locations on campus. We then "accidentally" format the camera's CompactFlash card, at which point we make a "forensic image" of the same (using `dd`) and beg students to recover as many of its JPEGs as they can by writing C code. The problem set then evolves into a course-wide scavenger hunt, as students are also enticed (as via pizza) to find the photos' locations on campus (and photograph themselves there).

Problem Set 6 fosters a friendly competition among students, who are handed a dictionary with 143,091 words and challenged to implement the fastest spell-checker possible (that uses minimal memory). Students (who opt in) are then ranked on the course's homepage according to their code's runtime. Students report spending more time on this problem set than any other, but only because so many are determined to out-do their friends.

Problem Set 7 has students implement, in PHP and SQL, an E*Trade-like site for "buying" and "selling" stocks. Thanks to freely available stock quotes from Yahoo Finance, students integrate actual prices (via CSV feeds) into their own sites.

Problem Set 8, finally, challenges students to implement a "mashup" using tiles from Google Maps and RSS feeds from Google News. Not only does this last project serve as a stepping stone to similarly Web-based final projects, it exposes them to yet another wonderfully documented API, this one from Google.

6. RESULTS

Not only has CS50's enrollment grown by 156% in three years, so have courses downstream grown in size too. Between 2006 and 2007 did CS51 (a LISP-based follow-up to CS50) almost double in size. Similarly did subsequent courses in theory increase their ranks, one by 33% and another by 122%. So has CS1, an optional lead-in to CS50, nearly tripled in size, from 31 to 90 this past year. To be fair, some of these same courses' enrollments have since dipped, though the department has begun offering many more next steps for students emerging from CS50, so we actually now "compete" with ourselves for students some terms.

Whether or not higher enrollments in CS50 and these other courses translates to more majors remains to be seen. Majors tend to take CS50 in their freshman or sophomore year, so a year or two hence will we have a better sense of any such trend.

7. CONCLUSION

In Fall 2007, we set out to reinvent CS50. Not only had the course suffered for years from a fear factor on campus, its problem sets were no longer consistent with topics genuinely of interest to students. And so we aspired to right both, while still preserving the course's historical rigor and underlying fundamentals. Two years' of end-of-term surveys indicate that we have indeed preserved the former, and careful attention to the course's curriculum has ensured that we have preserved the latter as well.

As the course's sheer size now attests, we appear to have transformed what was once, for many students, a course not to be taken into one that must be. Undoubtedly that mindset will not persist forever, as enrollments do seem to ebb and flow. But, for now, many more students do seem excited by computer science on campus.

APPENDIX

The address of CS50's website is:
`http://www.cs50.net/`

8. REFERENCES

[1] O. Astrachan and D. Reed. AAA and CS 1: The Applied Apprenticeship Approach to CS 1. In *SIGCSE '95: Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, pages 1–5, New York, NY, USA, 1995. ACM.

[2] O. Astrachan, R. Smith, and J. Wilkes. Application-Based Modules Using Apprentice Learning for CS 2. *SIGCSE Bull.*, 29(1):233–237, 1997.

[3] S. Cooper, W. Dann, and R. Pausch. Teaching objects-first in introductory computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 191–195, New York, NY, USA, 2003. ACM.

[4] David J. Malan. Virtualizing Office Hours in CS 50. *SIGCSE Bull.*, 41(3):303–307, 2009.

[5] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by Choice: Urban Youth Learning Programming with Scratch. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 367–371, New York, NY, USA, 2008. ACM.

[6] Pet Holdings, Inc. I Can Has Cheezburger? http://icanhascheezburger.com/.

[7] Stuart Zweben. 2007–2008 Taulbee Survey. Technical Report 3, Computing Research News, 2009.